HILL CIPHER

```
#include <stdio.h>
#include <string.h>

const int ALPHABET_SIZE = 26;
const char ALPHABET[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

void encrypt(char plaintext[], int keyMatrix[][3], char ciphertext[]) {
    int plaintextLength = strlen(plaintext);
    if (plaintextLength % 3 != 0) {
        printf("Plaintext length must be a multiple of 3\n");
        return;
    }

    for (int i = 0; i < plaintextLength; i += 3) {
        int plaintextVector[3];
        for (int j = 0; j < 3; j++) {
            plaintextVector[j] = plaintext[i + j] - 65;
        }

        int ciphertextVector[3];
        for (int i = 0; i < 3; i++) {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum += (keyMatrix[i][j] * plaintextVector[j]);
            }
            ciphertextVector[i] = sum % ALPHABET_SIZE;
        }

        for (int j = 0; j < 3; j++) {
            ciphertext[i + j] = ALPHABET[ciphertextVector[j]];
        }
    }
}

void decrypt(char ciphertext[], int inverseKeyMatrix[][3], char plaintext[]) {
    int ciphertextLength = strlen(ciphertext);
    if (ciphertextLength % 3 != 0) {
        printf("Ciphertext length must be a multiple of 3\n");
        return;
    }

    for (int i = 0; i < ciphertextLength; i += 3) {
        int ciphertextVector[3];
        for (int j = 0; j < 3; j++) {
            ciphertextVector[j] = ciphertext[i + j] - 65;
        }

        int plaintextVector[3];
        for (int i = 0; i < 3; i++) {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum += (inverseKeyMatrix[i][j] * ciphertextVector[j]);
            }
            plaintextVector[i] = sum % ALPHABET_SIZE;
        }

        for (int j = 0; j < 3; j++) {
            plaintext[i + j] = ALPHABET[plaintextVector[j]];
        }
    }
}

int main() {
    int keyMatrix[][3] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

    int inverseKeyMatrix[][3] = {{8, 5, 10}, {21, 8, 21}, {21, 12, 8}};

    char plaintext[] = "HELLO";
    char encryptedText[strlen(plaintext) + 1];
    encrypt(plaintext, keyMatrix, encryptedText);
    printf("Encrypted Text: %s\n", encryptedText);

    char decryptedText[strlen(plaintext) + 1];
    decrypt(encryptedText, inverseKeyMatrix, decryptedText);
    printf("Decrypted Text: %s\n", decryptedText);

    return 0;
}
```

DES

```
#include <stdio.h>
#include <conio.h>

/* Function to perform XOR operation between two arrays */
void xorOperation(int a[], int b[], int result[], int size) {
   for (int i = 0; i < size; i++) {
      result[i] = a[i] ^ b[i];
   }
}

/* Function to perform the encryption process */
void encrypt(int plaintext[], int key[], int encryptedText[], int size) {
   // Split the plaintext into left-hand and right-hand data
   int leftHand[size / 2];
   int rightHand[size / 2];

   for (int i = 0; i < size / 2; i++) {
      leftHand[i] = plaintext[i];
      rightHand[i] = plaintext[i + size / 2];
   }

   // Perform the first XOR operation with the key bit stream
   int firstXORData[size / 2];
   xorOperation(rightHand, key, firstXORData, size / 2);

   // Perform the second XOR operation with the left-hand data
   int secondXORData[size / 2];
   xorOperation(leftHand, firstXORData, secondXORData, size / 2);

   // Combine the encrypted left-hand and right-hand data
   for (int i = 0; i < size / 2; i++) {
      encryptedText[i] = secondXORData[i];
      encryptedText[i + size / 2] = rightHand[i];
   }
}

/* Main function */
int main() {
   // Initialize variables
   int plaintext[20], key[20], encryptedText[20];
   int n, k, i;

   // Get input from the user
   printf("\nEnter the plaintext number: ");
   scanf("%d", &n);

   printf("\nEnter the key number: ");
   scanf("%d", &k);

   printf("\nEnter the plaintext data: ");
```

```
    for (i = 0; i < n; i++) {
        scanf("%d", &plaintext[i]);
    }

    printf("\nEnter the key bit stream: ");
    for (i = 0; i < k; i++) {
        scanf("%d", &key[i]);
    }

    // Perform the encryption process
    encrypt(plaintext, key, encryptedText, n);

    // Display the encrypted data
    printf("\nEncrypted data: ");
    for (i = 0; i < n; i++) {
        printf("%d", encryptedText[i]);
    }

    getch(); // Pause before exiting

    return 0;
}
```

RSA ALGORITHM

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>RSA Encryption Demo</title>
  <script src="script.js"></script>
</head>
<body>
  <label for="plaintext">Plaintext:</label>
  <input type="text" id="plaintext" placeholder="Enter your message">
  <button onclick="encrypt()">Encrypt</button>
  <br>
  <label for="ciphertext">Ciphertext:</label>
  <input type="text" id="ciphertext" readonly>
</body>
</html>
```

JAVA SCRIPT

```
const jsbn = require('jsbn');

const keyPair = jsbn.generateKeys();
const publicKey = keyPair.publicKey;
const privateKey = keyPair.privateKey;

console.log('Public Key:', publicKey);
console.log('Private Key:', privateKey);

var message = document.getElementById('plaintext').value;
const ciphertext = rsaEncrypt(message, publicKey);
console.log('Encrypted Message:', ciphertext);



function rsaEncrypt(message, publicKey) {
  // Convert message to a BigInt
  const messageBigInt = BigInt(message);

  const ciphertext = messageBigInt.pow(publicKey.e) % publicKey.n;

  document.getElementById('ciphertext').value = ciphertext;
}
```

SHA-1

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA1 {

    public static String sha1(String text) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] bytes = md.digest(text.getBytes());

        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            hexString.append("%02x", b);
        }

        return hexString.toString();
    }

    public static void main(String[] args) throws NoSuchAlgorithmException {
        String text = "Hello, world!";
        String hash = sha1(text);
        System.out.println(hash);
    }
}
```

DSS

```
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;

public class DigitalSignature {

    public static void generateSignature(String message, PrivateKey privateKey) throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        signature.update(message.getBytes());
        byte[] signatureBytes = signature.sign();

        System.out.println("Generated Signature: " + bytesToHex(signatureBytes));
    }

    public static boolean verifySignature(String message, byte[] signatureBytes, PublicKey
publicKey) throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initVerify(publicKey);
        signature.update(message.getBytes());
        return signature.verify(signatureBytes);
    }

    public static void main(String[] args) throws Exception {
        // Generate key pair
        KeyPair keyPair = KeyPairGenerator.getInstance("RSA").generateKeyPair();
        PrivateKey privateKey = keyPair.getPrivate();
        PublicKey publicKey = keyPair.getPublic();

        // Generate message
        String message = "Hello, world!";

        // Generate signature
        generateSignature(message, privateKey);

        // Verify signature
        boolean verified = verifySignature(message, signatureBytes, publicKey);
        System.out.println("Signature verification: " + verified);
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            hexString.append(String.format("%02x", b));
        }
        return hexString.toString();
    }
}
```

DIFFEE HELLMAN

```
import java.math.BigInteger;

public class DiffieHellman {

    public static void main(String[] args) {
        // Choose a prime number p and a primitive root g of p
        BigInteger p = new BigInteger("11"); // Prime number
        BigInteger g = new BigInteger("2");  // Primitive root of p

        // Alice and Bob choose private keys a and b
        BigInteger a = new BigInteger("3");  // Alice's private key
        BigInteger b = new BigInteger("7");  // Bob's private key

        // Alice computes A = g^a mod p
        BigInteger A = g.modPow(a, p);

        // Bob computes B = g^b mod p
        BigInteger B = g.modPow(b, p);

        // Alice sends A to Bob
        System.out.println("Alice sends A to Bob: " + A);

        // Bob sends B to Alice
        System.out.println("Bob sends B to Alice: " + B);

        // Alice computes the shared secret key S = B^a mod p
        BigInteger sharedSecretA = B.modPow(a, p);

        // Bob computes the shared secret key S = A^b mod p
        BigInteger sharedSecretB = A.modPow(b, p);

        // Verify that the shared secret keys are equal
        if (sharedSecretA.equals(sharedSecretB)) {
            System.out.println("Shared secret key: " + sharedSecretA);
        } else {
            System.out.println("Error: Shared secret keys are not equal.");
        }
    }
}
```

TRANSPOSITION CIPHER

```
import java.util.Arrays;

public class TranspositionCipher {

  public static void main(String[] args) {
    // Plaintext message
    String plaintext = "Hello, World!";

    // Row transformation
    char[][] matrix = new char[3][4];
    int k = 0;
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 4; j++) {
        matrix[i][j] = plaintext.charAt(k++);
      }
    }

    // Column transformation
    char[] ciphertext = new char[12];
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 3; j++) {
        ciphertext[i * 3 + j] = matrix[j][i];
      }
    }

    // Print ciphertext
    System.out.println("Ciphertext: " + new String(ciphertext));
  }
}
```

CAESAR CIPHER

```
import java.util.Scanner;
public class CaesarCipher {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get plaintext message
        System.out.print("Enter plaintext message: ");
        String plaintext = scanner.nextLine();

        // Get shift value
        System.out.print("Enter shift value: ");
        int shift = scanner.nextInt();

        // Encrypt the plaintext message
        String ciphertext = encrypt(plaintext, shift);
        System.out.println("Encrypted message: " + ciphertext);

        // Decrypt the ciphertext
        String decryptedText = decrypt(ciphertext, shift);
        System.out.println("Decrypted message: " + decryptedText);
    }

    private static String encrypt(String plaintext, int shift) {
        char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
        StringBuilder ciphertext = new StringBuilder();

        for (char c : plaintext.toCharArray()) {
            int index = alphabet.indexOf(c);
            int newIndex = (index + shift) % alphabet.length;
            ciphertext.append(alphabet[newIndex]);
        }

        return ciphertext.toString();
    }

    private static String decrypt(String ciphertext, int shift) {
        char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
        StringBuilder decryptedText = new StringBuilder();

        for (char c : ciphertext.toCharArray()) {
            int index = alphabet.indexOf(c);
            int newIndex = (index - shift) % alphabet.length;
            if (newIndex < 0) {
                newIndex += alphabet.length;
            }
            decryptedText.append(alphabet[newIndex]);
        }
        return decryptedText.toString();
    }
}
```

RAIL FENCE

```
import java.util.Scanner;

public class RailFenceCipher {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get plaintext message
        System.out.print("Enter plaintext message: ");
        String plaintext = scanner.nextLine();

        // Get number of rails
        System.out.print("Enter number of rails: ");
        int rails = scanner.nextInt();

        // Encrypt the plaintext message
        String ciphertext = encrypt(plaintext, rails);
        System.out.println("Encrypted message: " + ciphertext);

        // Decrypt the ciphertext
        String decryptedText = decrypt(ciphertext, rails);
        System.out.println("Decrypted message: " + decryptedText);
    }

    private static String encrypt(String plaintext, int rails) {
        char[][] matrix = new char[rails][plaintext.length()];
        int k = 0;
        boolean down = true;
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < plaintext.length(); j++) {
                matrix[i][j] = plaintext.charAt(k++);
            }
            down = !down;
        }

        StringBuilder ciphertext = new StringBuilder();
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < plaintext.length(); j++) {
                if (matrix[i][j] != 0) {
                    ciphertext.append(matrix[i][j]);
                }
            }
        }

        return ciphertext.toString();
    }

    private static String decrypt(String ciphertext, int rails) {
        int cols = ciphertext.length() / rails;
        char[][] matrix = new char[rails][cols];
        int k = 0;
        boolean down = true;
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < cols; j++) {
                if (down) {
                    matrix[i][j] = ciphertext.charAt(k++);
                } else {
                    matrix[rails - i - 1][cols - j - 1] = ciphertext.charAt(k++);
                }
            }
            down = !down;
        }

        StringBuilder decryptedText = new StringBuilder();
        for (int i = 0; i < plaintext.length(); i++) {
            for (int j = 0; j < rails; j++) {
                if (matrix[j][i] != 0) {
                    decryptedText.append(matrix[j][i]);
                }
            }
        }

        return decryptedText.toString();
    }
}
```

VIGNERE CIPHER

```
import java.util.Scanner;

public class VigenereCipher {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get plaintext message
        System.out.print("Enter plaintext message: ");
        String plaintext = scanner.nextLine();

        // Get keyword
        System.out.print("Enter keyword: ");
        String keyword = scanner.nextLine();

        // Encrypt the plaintext message
        String ciphertext = encrypt(plaintext, keyword);
        System.out.println("Encrypted message: " + ciphertext);

        // Decrypt the ciphertext
        String decryptedText = decrypt(ciphertext, keyword);
        System.out.println("Decrypted message: " + decryptedText);
    }

    private static String encrypt(String plaintext, String keyword) {
        char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
        char[] keywordChars = keyword.toLowerCase().toCharArray();
        int keywordLength = keywordChars.length;
        StringBuilder ciphertext = new StringBuilder();

        for (int i = 0, j = 0; i < plaintext.length(); i++) {
            char c = plaintext.charAt(i);
            if (c < 'A' || c > 'Z') {
                ciphertext.append(c);
                continue;
            }

            int plainIndex = alphabet.indexOf(c);
            int keywordIndex = alphabet.indexOf(keywordChars[j % keywordLength]);
            int shift = (keywordIndex + 26 - plainIndex) % 26;
            int newIndex = (plainIndex + shift) % 26;
            ciphertext.append(alphabet[newIndex]);

            j++;
        }

        return ciphertext.toString();
    }

    private static String decrypt(String ciphertext, String keyword) {
```

```
    char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
    char[] keywordChars = keyword.toLowerCase().toCharArray();
    int keywordLength = keywordChars.length;
    StringBuilder decryptedText = new StringBuilder();

    for (int i = 0, j = 0; i < ciphertext.length(); i++) {
      char c = ciphertext.charAt(i);
      if (c < 'A' || c > 'Z') {
        decryptedText.append(c);
        continue;
      }

      int cipherIndex = alphabet.indexOf(c);
      int keywordIndex = alphabet.indexOf(keywordChars[j % keywordLength]);
      int shift = (cipherIndex - keywordIndex + 26) % 26;
      int newIndex = (cipherIndex - shift) % 26;
      decryptedText.append(alphabet[newIndex]);

      j++;
    }

    return decryptedText.toString();
  }
}
```